



The Secret Life of Automation

Michael Bolton
michael@developsense.com
<http://www.developsense.com>
@michaelbolton

1

I'm Michael Bolton



Not the singer.



**Not the guy
in Office Space.**



No relation.

The Secret Life of Automation.pdf - 2

2

I'm Michael Bolton



I help people solve testing problems they didn't know they could solve.
And I help them learn how to do that for themselves.

The Secret Life of Automation.pdf - 3

3

**This talk is based on personal experience, surveys of
stuff on the web, and on interviews with friendly
and generous colleagues.**

Thank you to James Bach, Adam Goucher, Pete Houghton, Ben Simo,
Andy Tinkham, and Anonymous

The Secret Life of Automation.pdf - 4

4

This talk is based on personal experience, surveys of stuff on the web, and on interviews with friendly and generous colleagues.

Selection Biases

People who don't need my help don't ask me for help.

People who *really* need help don't ask *anyone* for help.

The Secret Life of Automation.pdf - 5

5

**This talk isn't going to work.
You're going to interpret it from your existing
perspective (or one you formed in a few seconds).**

However, we'll give it a try.

The Secret Life of Automation.pdf - 5

6

A Fable about the Roomba

ID 102881079 © Konstantinos A | Dreamstime.com



“We came home to find that the cats had crapped everywhere. While we were addressing what we discovered, I started up the Roomba. The Roomba smeared the cat shit all over the house.

“The lesson: you have to clean up your shit before you automate.”

Ben Simo

The Secret Life of Automation.pdf - 32

7

SSSSSHHHH!

success = commercial advantage
failure = embarrassment
current state = **unknown**

8

What is not being talked about?

what people do in “test automation” work
what people do in *all* testing work
what people do in development work
what machinery really does
how tools could *extend* skills
concepts and forms

9

**There's no sense in being precise when you
don't even know what you're talking about.**

John von Neumann

The Secret Life of Automation.pdf - 13

10

OK, so what *are* we talking about?

testing

evaluating a product by **learning** about it through **exploration** and **experimentation** and **experience**, which includes to some degree: **questioning**, **study**, **modeling**, **observation** and **inference**, including...

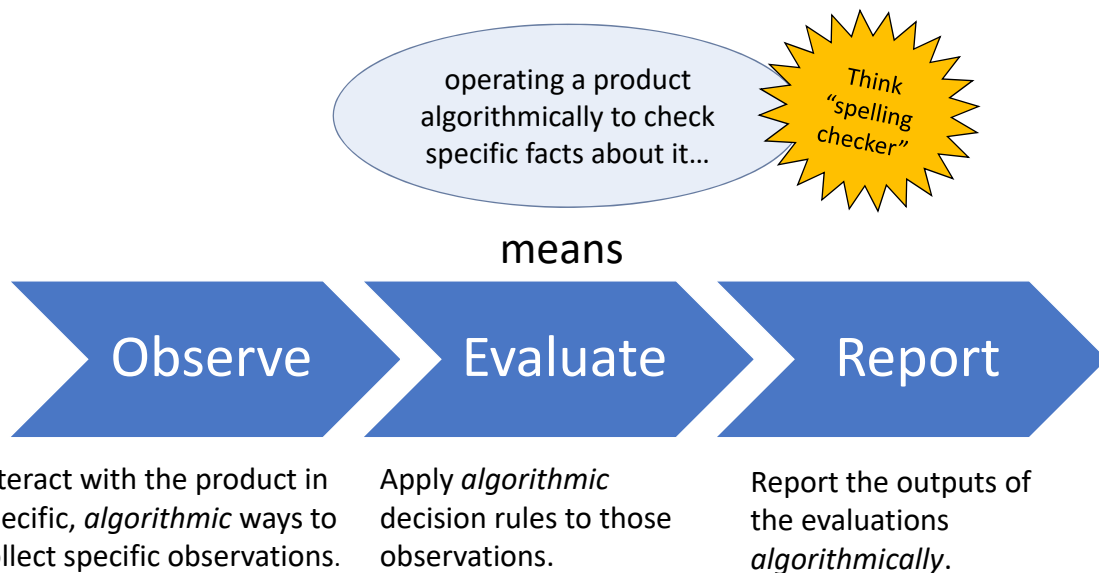
checking

the process of making evaluations by applying **algorithmic** decision rules to specific observations of a product

The Secret Life of Automation.pdf - 14

11

Call This “Checking”, Not Testing



The Secret Life of Automation.pdf - 15

12

OK, so what else are we talking about?

 automation n. “A high degree of mechanization in **manufacture**, the **handling of material** between processes being automatic and **the whole system** being automatically controlled.”

—*Chambers Dictionary (iOS)*

 automation n. “the **use** or introduction of automatic equipment in a **manufacturing** or other process or facility.”

—*Concise Oxford Dictionary*

The Secret Life of Automation.pdf - 23

13

(How about “tool”?)

 a working instrument, esp. one used by hand

 the cutting part of a machine tool

 someone who is used as the mere instrument of another

 anything necessary to the pursuit of a particular activity

 a fool (slang)

 a despicable person (slang)

 a utility, feature or function available as part of e.g. a word processing package or database (computing)

—*Chambers Dictionary (iOS)*

The Secret Life of Automation.pdf - 24

14

In Rapid Software Testing, we offer...



tool (n.)



any contrivance used to fulfill a human purpose



test tool (idiom)



any tool used in the service of testing



automation (n.)



1. any process *entirely* performed by a tool



2. a tool capable of such performance



exploratory testing (adj. + gerund)



testing

The Secret Life of Automation.pdf - 25

15

Secret:

Automated *testing* **does not exist.**

It *cannot* exist.

The Secret Life of Automation.pdf - 35

16

You said...

“Automate all the testing!”

You might have meant...

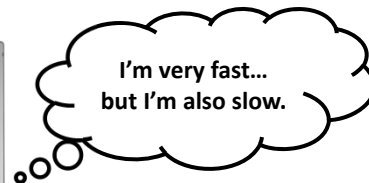
Automate all the checking

and risk assessment
and task prioritization
and which means
and pattern recognition
and decision making
and design of the test lab
and preparation of the test lab
and sensemaking
and test code development
and tool selection
and recruiting of helpers
and making test notes
and preparing simulations
and interacting with developers
and bug investigation
and problem triage
and relationship building
and product configuration
and application of oracles
and designing visualizations
and spontaneous playful interaction

17

Testing Is *More Than* Checking

- *Checking* is okay, but it is mostly focused on confirming what we know or hope to be true. And programming checks takes time.
- To understand our products and the risk of problems that matter to people, we must do more than output checking; we must *test*. And we must do that to design excellent checks, too.

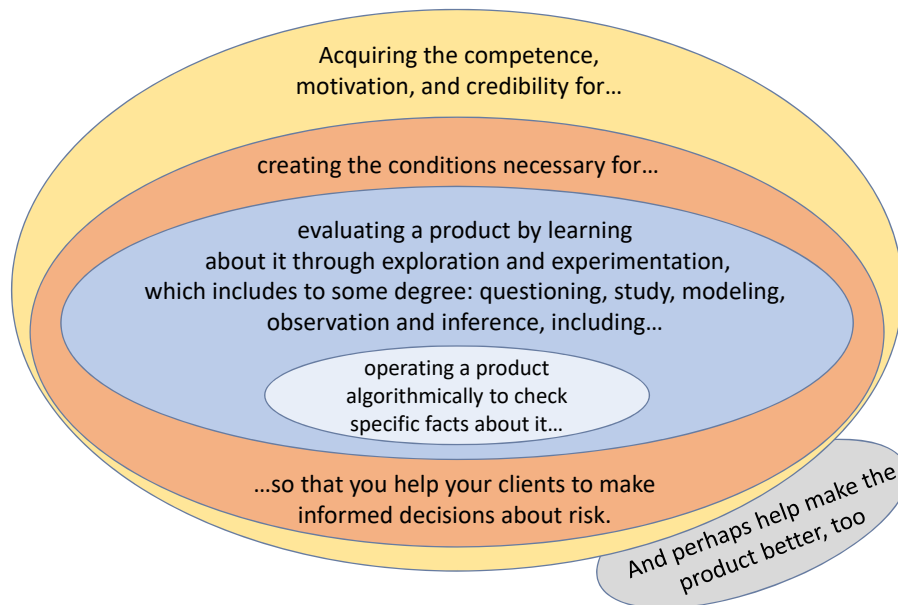


See <http://www.developsense.com/2009/08/testing-vs-checking.html>

The Secret Life of Automation.pdf - 16

18

Testing is...



19

Secret:
A test cannot be automated.

The Secret Life of Automation.pdf - 33

20

**Also Not Exactly a Secret:
If we keep talking about automated
testing, naïve people will continue to
believe that testing can be automated.**

The Secret Life of Automation.pdf - 38

21

Why it's important to distinguish testing and checking

- Because *checking* is mechanistic. It can be made completely **explicit** and automated. It is *inside* testing. It is a *tactic* of testing.



People don't
confuse biting
with eating!

Biting can be
done by
tools... but
eating can't.



The Secret Life of Automation.pdf - 18

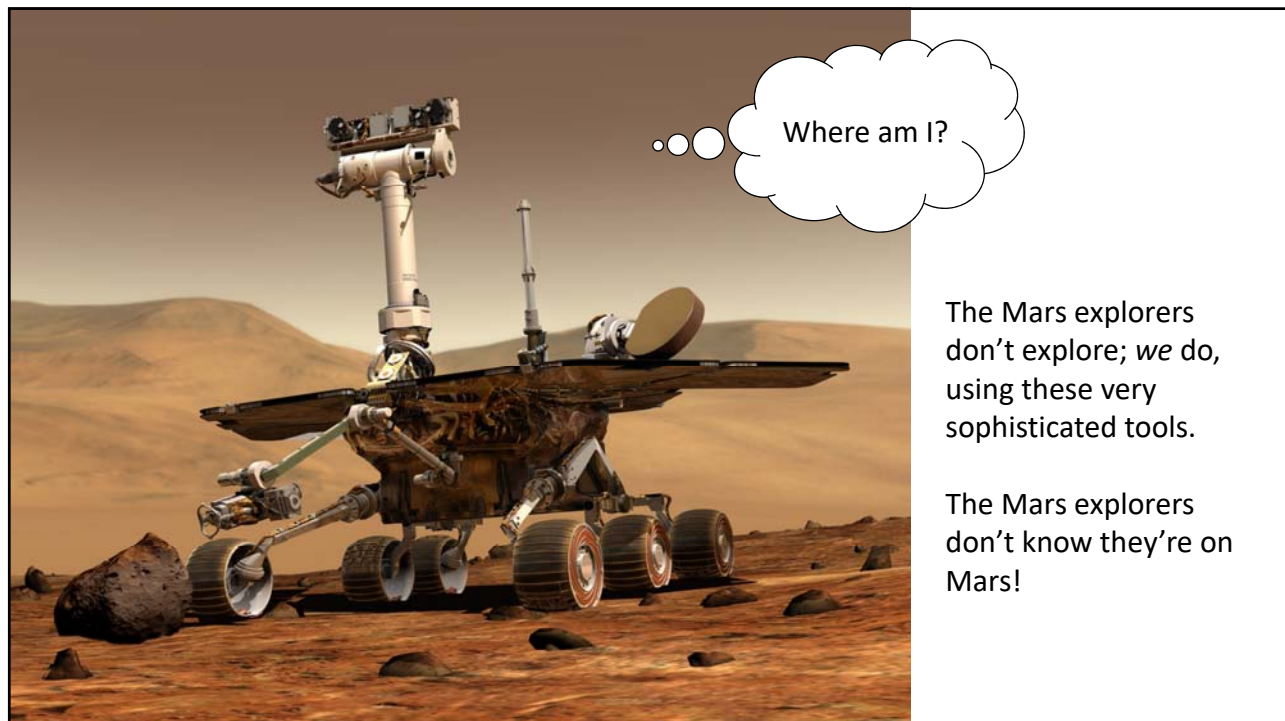
22

Why it's important to distinguish testing and checking

- Because *checking* is mechanistic. It can be made completely **explicit**, encoded, and automated. It is *inside* testing. It is a *tactic* of testing.
- Because *testing* involves **tacit** and **social skills** that cannot be encoded. Testing skills must be developed through socialization, practice, and increasingly challenging work, not via rote procedures.
- Because talk about efficiency and effectiveness makes for *very* different conversations when we're talking about explicit vs. tacit skills.
- Because for checking to be *truly* excellent, it must be embedded in excellent testing. Developing valuable checks requires skill!
- Programmers have resisted marginalization for years!
(They no longer call compilers "autocoders" and programming languages are no longer called "autocodes".)

The Secret Life of Automation.pdf - 19

23

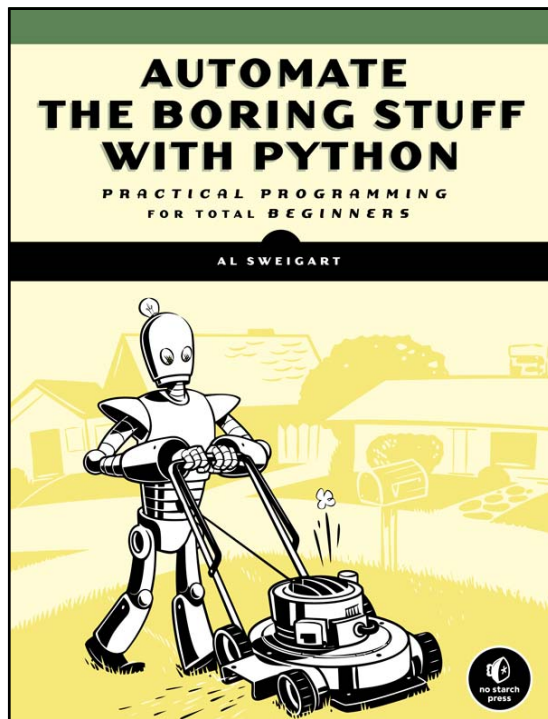


24

Secrets (things we're not talking about):

- Machines don't understand context. Only humans do.

25



What can we say about this cartoon?

- The robot is almost certainly electrically powered, but the lawnmower is set up with a gas engine.
- The robot is humanoid, using a lawnmower's human interface. Why?
- If you wanted to automate the process of mowing the lawn, why not make things simpler and get rid of both the humanoid robot *and* the human user interface?
- If there are obstacles on the lawn, we can't see them.
- There's no bag to collect the cuttings, so depending on what you want, there may be some raking to do.
- There are no *people* in the picture. There are no sidewalks, either.

But let's acknowledge that the cartoon is *not to be taken seriously*. It's whimsical and silly. That's OK. The trouble is, we often seen automation in testing considered in this cartoonish way.

The Secret Life of Automation.pdf - 27

26



Photo source: <https://www.amazon.com/Robomow-RS622-Battery-Operated-Mower/dp/B00IGXYBFI>

This is a much more sensible design, but we're not out of the woods yet.

In order for the mower to understand where it's supposed to go, it needs a sensing wire to be put down around the edges of the lawn, and around any obstacles inside the area. The reviews are decidedly mixed; half are five stars, while one-quarter of them give only one star.

It's three times as expensive as a push mower (and this model is about half the price of models with a better rating). Putting down a boundary wire makes sense if your lawn and the obstacles in it don't change. If you have a huge space to cover, and you're not too fussy about how it gets covered, the robot mower might make a lot of sense. But the real point is that software isn't much like this anyway.

The Secret Life of Automation.pdf - 29

29

Here is one buyer's one-star review of the product.

My robot came with software version 0.86, it could not drive in straight lines, and it would randomly just freak out and say "not in working area", or "upside down" when neither was the case. By the way, the robot will think it has launched into the air and flipped over if it hits a tiny bump in my lawn. You literally need to have no bumps in your yard.

Eventually he changed his rating from one to three stars after the manufacturer shipped him a replacement for his first robot. He was better satisfied, but included these caveats: *This robot will not mow your whole lawn. Over the summer, I've had to slowly cede areas that the robot just can't handle. There were already a few areas like this, but I've had to add more and more area to the manual mow list. Sometimes something as simple as a tiny twig after fallen a rain will completely alter the robot's course over your lawn. It might cause the robot to get stuck where it never previously did get stuck.*

After 11 months, he provide one more update of the robot that he had come to call "Larry". *Larry started misbehaving - sometimes he'd drive in circles or just change direction at random and then start behaving normally. I didn't think much of it since I'd checked the boundary wire, the light on the base was green and he seemed to pull himself out of his funk after a minute or two.*

UNTIL, it's everyone's nightmare - a few days later you come home and find a body floating in the pool. LARRY!!!

30

Secrets (things we're not talking about):

- Machines don't understand context. Only humans do.
- We test to investigate possible problems in the product. But people (especially builders) don't like to talk about problems!

31

Secrets (things we're not talking about):

- Machines don't understand context. Only humans do.
- We test to investigate possible problems in the product. But people (especially builders) don't like to talk about problems!
- Human interfaces are a poor match for mechanical users. BUT automating the human interface is intriguing and somewhat dazzling.

32

Now, for the new word...

GEMPUB

The Secret Life of Automation.pdf - 45

33

Here it is in a bunch of different fonts...

GEMPUB
GEMPUB
GEMPUB
GEMPUB
GEMPUB
GEMPUB

The Secret Life of Automation.pdf - 46

34

What does this ugly, stupid word mean?

- **GETting**
- **M**achines to
- **P**Ush
- **B**uttons



GEMPUB (n.) getting machines to push buttons

The Secret Life of Automation.pdf - 47

35

Why the fixation on GEMPUB?

Popular test framing pattern #1

1. Testing is misconceived as being about *confirming that the product works*; checking.
2. There is a large test space to cover.
3. This space is currently being covered by (shallow) “manual test cases” (i.e., human checking).
4. These checks took a long time to develop and write up, representing a sunk cost, so we don’t want to throw them out.
5. Since they’re already pretty close to being algorithmic, we can reuse them (even though they’re not very valuable).

The Secret Life of Automation.pdf - 48

36

Why the fixation on GEMPUB?

Popular test framing pattern #2

1. Testing is misconceived as being about *confirming that the product works*; checking.
2. There is a large test space to cover.
3. People want *something* done to prevent total embarrassment (e.g. a missing screen)
4. Therefore, they'll often settle for showing that something exists and that it *can* work.
5. Getting a tool to *visit* every screen *sounds* cheap, and it's worth *something*.
6. As long as we can do that, deeper bugs are simply bad luck; too hard to find.

The Secret Life of Automation.pdf - 49

37

Secrets (things we're not talking about):

- Machines don't understand context. Only humans do.
- We test to investigate possible problems in the product. But people (especially builders) don't like to talk about problems!
- Human interfaces are a poor match for mechanical users. BUT automating the human interface is intriguing and somewhat dazzling.
- People can relate to button-pushing as slow, repetitive, and tedious, so with GEMPUB, you can look busy AND dazzle the client.

38

What I have learned from conferences, books, blogs, articles, and testing forums

The most thoroughly tested part of any application is

You are now logged in.

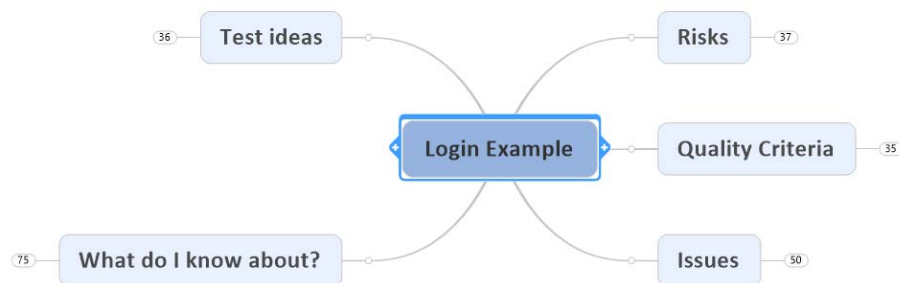
And thanks to GEMPUB, machines can log in
hundreds of times a minute.

And people will say “Lo! There be testing!”

The Secret Life of Automation.pdf - 40

39

Login Stuff We MIGHT Want To Test



The Secret Life of Automation.pdf - 41

40

What I **haven't** learned about from conferences, books, blogs, articles, and testing forums

- risk (referring to the product)
- problem (referring to the product)
- bug, error, defect, etc. (referring to the product)
- quality
- value
- coverage
- oracles
- investigation
- discovery
- learning

The Secret Life of Automation.pdf - 42

41

Analysis at a client site...

- Client had 1100 automated checks, developed over several years mostly by outside contractors.
- These checks took approximately 24 hours(!) to run.
- Of these 1100, 100 were regularly running red.
- Of those 100, 25 were known environmental problems (therefore considered non-bugs).
- Of the remaining 75, about 10% (~7 total) were regularly false-positive reports (that is, non-bugs).

What questions would you ask?

The Secret Life of Automation.pdf - 54

42

Questions from My Analysis

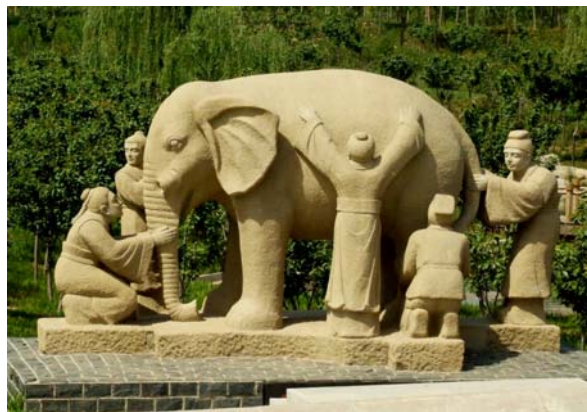
- ? What are these checks actually checking?
- ? When you say “1100 checks”, is each check examining a single condition, or multiple conditions?
- ? Might it be more valuable to talk about coverage, rather than counting checks?
- ? Do the checks include variation to expose bugs?
- ? How often are the checks reviewed and analyzed for relevance and accuracy?
- ? Are the checks duplicating conditions already checked by programmers?
- ? Are the programmers doing *any* of their own checking?
- ? Why are the checks taking 24 hours?!
- ? What role does setup play in the timing?

The Secret Life of Automation.pdf - 55

43

Plus the invisible elephant...

- If 7% of the “failing” checks were unreliable, why presume that the “passing” checks were 100% reliable?



The Secret Life of Automation.pdf - 56

44

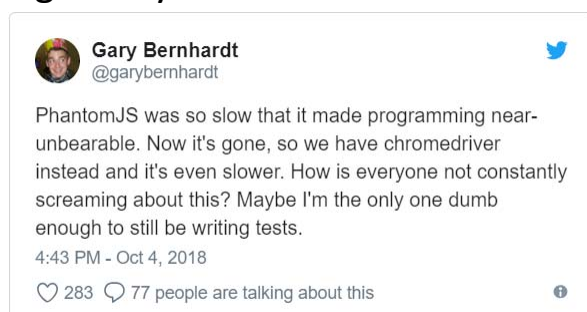
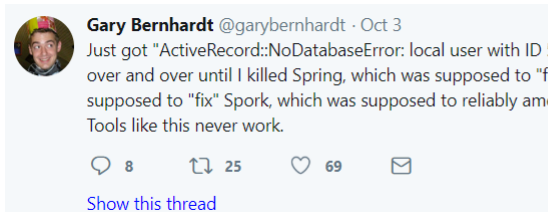
Secrets (things we're not talking about):

- If you're not a great risk investigator and you're not a great programmer, you can make yourself look busy with GEMPUB as long as no one looks too closely.
- GEMPUB can create enough output that no one questions any check unless it reports a problem.

45

We shape our tools, and thereafter our tools ~~shape~~ **screw** us.

- Preparing for a class, I wanted to do a checking demo.
- I innocently installed Cucumber.
- This completely clobbered my Selenium installation.
- I troubleshot. I hacked.
- Before I knew it, four hours had gone by.



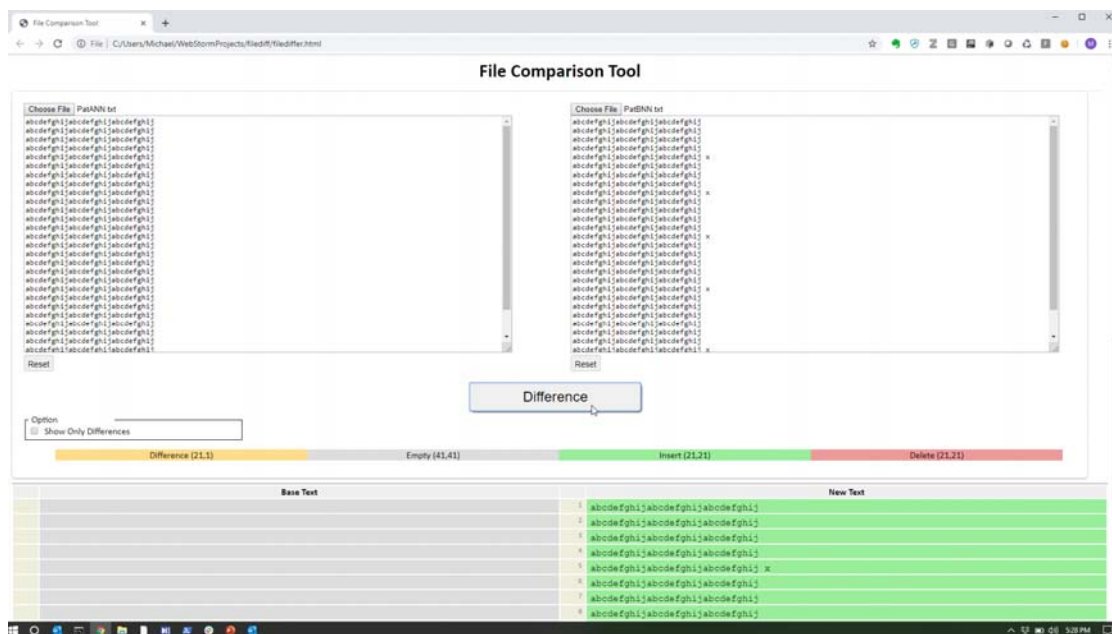
46

Secrets (things we're not talking about):

- If you're not a great risk investigator and you're not a great programmer, you can make yourself look busy with GEMPUB as long as no one looks too closely.
- GEMPUB can create enough output that no one questions any check unless it reports a problem.
- Since we're supposed to be capable problem-solvers, we're usually disinclined to admit how fiddly our tools are, and how much time we spend futzing with them.

47

Once Upon A Time...



48

In 2018, James Bach and I began to develop an exercise centred on FileDiffer, a single-page Web app that compare text files. We wanted such an app such that (among other things) people could manipulate and query its UI via tools (Ruby, Selenium, Excel).

I've worked professionally as a programmer in the past. But these days I don't consider myself a practicing programmer; I don't write test code every day. Without daily practice, I get rusty, and forget little details and nuances of the languages and the frameworks... even ones that I've written myself.

I wanted to produce examples of tools and automated checks that people might use to help test FileDiffer, using both good and not-so-good approaches. One example of a not-so-good approach is to rush into automating the product's behaviour through the GUI, before studying the product and learning about it.

So, I started with opening the browser and programming a couple of routines to find the input fields and the buttons, with the goal of producing totally simple checks and then extending them. Yay! — fast progress. I did one simple check of two chunks of text with NO differences, and another check for a really simple difference.

The Secret Life of Automation.pdf - 7

49

At several points I ran into testability-related bugs in the product. For instance, there are two Reset buttons in the UI, and both had the same element ID. I spent several minutes pondering and researching how to get around that problem (I wanted to be faithful to the scenario and not modify the source code).

I hacked away at studying aspects of XPath. Later I found out that James had fixed that bug in a new version of the product code. This happens in real projects! Often product code is being developed concurrently with the check code, and it's easy for the two paths of development to get out of sync.

It took me a couple of fiddly hours to get some check code running the way I wanted it to. No doubt if I were in better practice, that would have gone down to an hour or so.

Here's the bad part, though. At one point, the product was giving me output that I found surprising and confusing. Rather than investigating what was going on, I ignored my confusion. Instead, *I fixed the test code to match the product's output.*

The Secret Life of Automation.pdf - 8

50

I stepped into the Green Bar Trap!

While preparing automated checks for the File Differ program that we use in our Rapid Software Testing classes, I got results that I found confusing.

I caught myself altering the predicted, desired results of checks *based on what the program was doing*. This might have been fine for regression testing if the product had been working properly.

I was avoiding learning how the algorithm actually behaved! It took an *embarrassing* amount of time to catch myself at this.

(Secret: there are *significant* bugs in the algorithm, bugs that I would have found much earlier if I had confronted confusion instead of running from it. We found these bugs by using realistic, representative data; and we found more by generating volumes of data with interesting patterns, using tools.)

Developing an understanding of a product can be *hard*, and the temptation to **JUST MAKE THE CHECK CODE RUN GREEN** can be *overwhelming*, even for someone who famously warns people against exactly that.

The Secret Life of Automation.pdf - 9

51

Early Engagement

- When you first encounter the product, your ideas about what to check will be pretty simplistic, and probably not so valuable.
 - Your ideas will probably be oriented towards demonstration, and less towards investigation (yes, checking can be used in an investigative, exploratory way).
- Especially during early days, engage in *testing to learn*. Explore the product and experiment with it.
- If it helps, consider writing throwaway checks. But don't formalize too much too soon.
- Whatever risks you don't think of, your checks won't think of them either. So at first, focus on learning about the product and risks about it.

Higher-Value Checking - 10

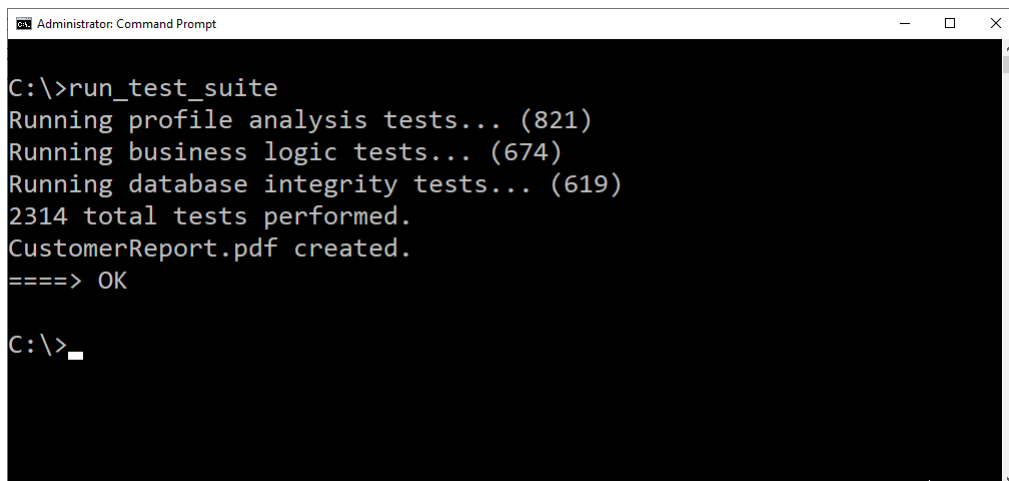
52

Premature Checking

- Once upon a time, OrgB Technologies set up automated checks to confirm that a PDF report was being created. Checks showed that PDFs *were* being created.

The Secret Life of Automation.pdf - 59

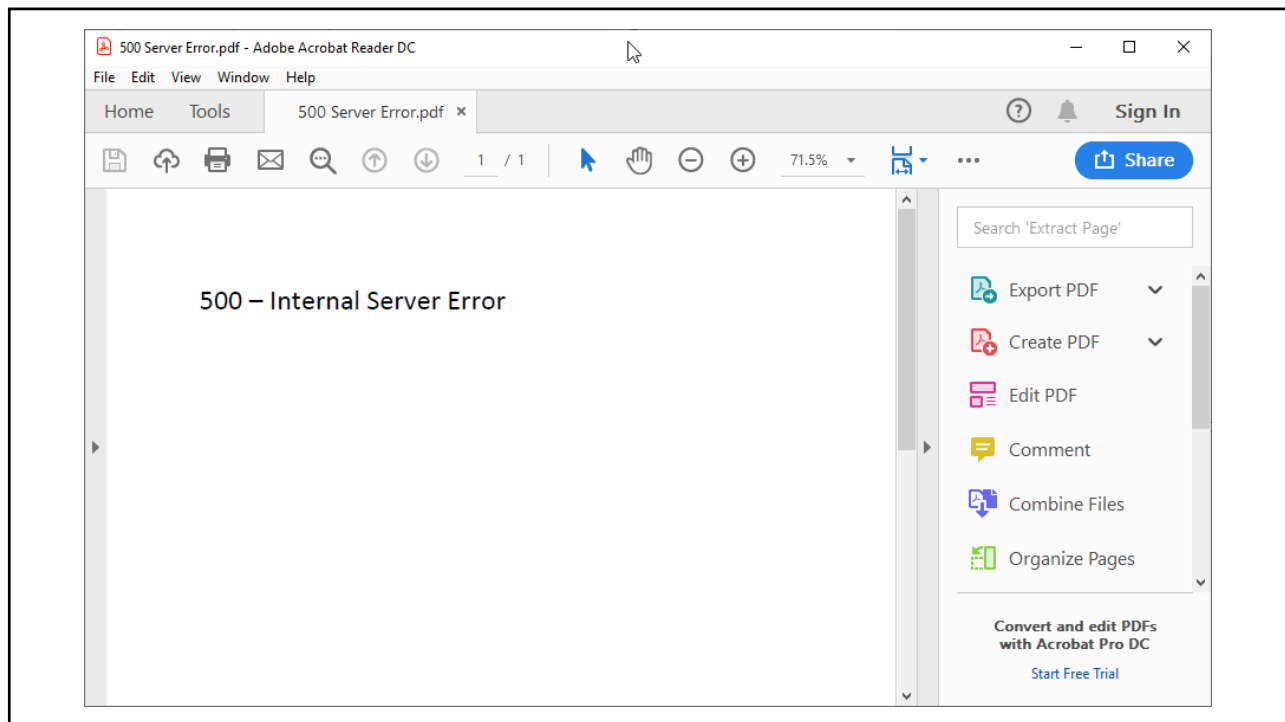
53



```
Administrator: Command Prompt
C:\>run_test_suite
Running profile analysis tests... (821)
Running business logic tests... (674)
Running database integrity tests... (619)
2314 total tests performed.
CustomerReport.pdf created.
====> OK

C:\>_
```

54



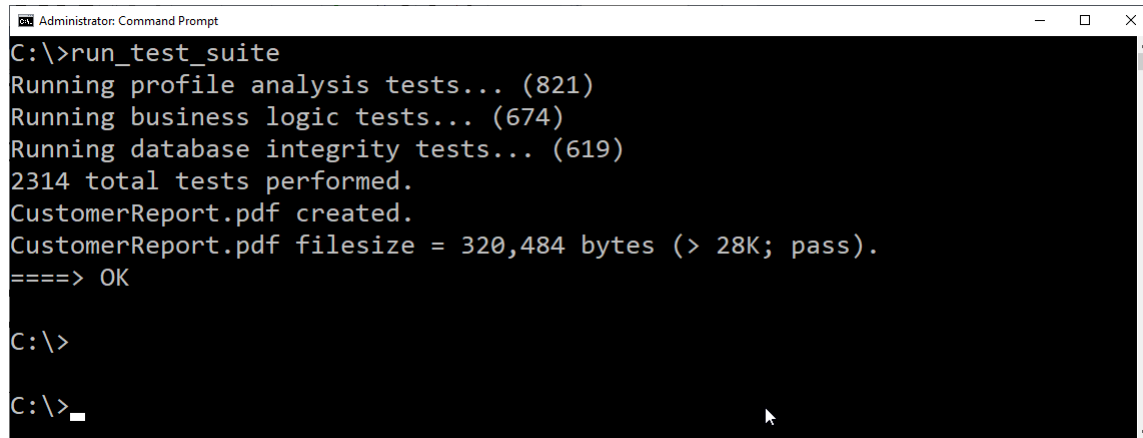
55

Premature Checking

- Once upon a time, OrgB Technologies set up automated checks to confirm that a PDF report was being created. Checks showed that PDFs *were* being created.
- Testers added checks to confirm that the PDF was the right size (the PDF spec was, at that time, proprietary and not available).

The Secret Life of Automation.pdf - 59

56



```
Administrator: Command Prompt
C:\>run_test_suite
Running profile analysis tests... (821)
Running business logic tests... (674)
Running database integrity tests... (619)
2314 total tests performed.
CustomerReport.pdf created.
CustomerReport.pdf filesize = 320,484 bytes (> 28K; pass).
====> OK

C:\>

C:\>_
```

57

Premature Checking

- Once upon a time, OrgB Technologies set up automated checks to confirm that a PDF report was being created. Checks showed that PDFs *were* being created.
- Testers added checks to confirm that the PDF was the right size (the PDF spec was, at that time, proprietary and not available).
- Somewhat later, OrgB discovered that although the PDFs were about the right size, certain columns were not being included. So, after some time, OrgB found a library that was able to read elements of PDF files. This made it straightforward to determine that the right number of columns were in the file.
- Somewhat later, OrgB discovered that although the columns were being included, they were disappearing off the right-hand edge of the page.
- Somewhat later, OrgB discovered that the now visible columns contained invalid data...

The Secret Life of Automation.pdf - 59

58

More Secrets (things we're not talking about):

- To test a product well, you need to get experience with it. To get *valuable* experience with it, you have to interact with it.
- To get to retesting or regression testing, you must do new testing first.
- GEMPUB is a very shallow form of interaction with the product, even when you have to debug “failing” checks. Once you’ve fixed the bugs in the check code, it’s easy to go back to sleep.
- GEMPUB might have some usefulness for regression checking. But developers’ integration and unit checks might be more useful by being more directly targeted. Meanwhile, testers need to engage with the product.

59

Healthy Perspective About Checks

We like checks. We use checks. They can help us affirm that the product can work, or if the product has suddenly stopped working in ways that are covered by the checks. Here are some caveats, though:

- Testing requires *learning*, and checks can’t learn.
- Demonstrating that a product **can** work is far from showing that it **will** work under various kinds of challenges.
- The performance of a check can be automated. But designing, programming, interpreting, and improving checks—that is, the testing work that *surrounds* checking—*can’t* be automated.

The Secret Life of Automation.pdf - 20

60

The Secret Life of Automation

Testing, by and large, exists in a constant state of existential crisis — at least at places where it is deemed overhead, wasteful or a cost. So we choose to automate—but what, precisely are we automating? Is it only GEMPUB?

The Secret Life of Automation.pdf - 68

61

Final Thoughts

- Testing is not only about functionality or technical correctness. Those things are important to some degree.
- But testing is really about learning. It's about discovering the fit (and lack of fit) between software systems and the people who use and develop them.
- Testing is not about confirming that the product CAN work. That's a *building* thing. It might be important, but it's not *testing*.
- Testing is about *challenging* the product and finding problems.
- In order to do that, the tester must change.

The Secret Life of Automation.pdf - 10

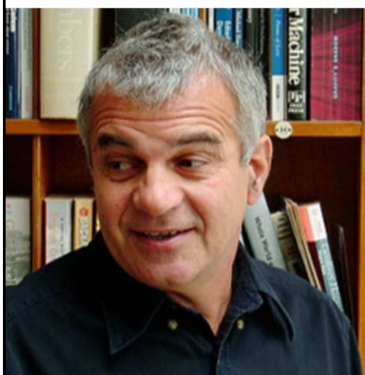
62

For Readers Only: Secret Secrets

The Secret Life of Automation.pdf - 69

63

Testing Is *Social Science*



Harry Collins
Abstract:
"Machines as Social Prostheses"
EuroSTAR 2013

"Computers and their software are two things. As collections of interacting cogs they must be 'checked' to make sure there are no missing teeth and the wheels spin together nicely.

Machines are also 'social prostheses', fitting into social life where a human once fitted. It is a characteristic of medical prostheses, like replacement hearts, that they do not do exactly the same job as the thing they replace; the surrounding body compensates.

The Secret Life of Automation.pdf - 21

64

Testing Is *Social Science*



Harry Collins
Abstract:
“Machines as Social Prostheses”
EuroSTAR 2013

“Contemporary computers cannot do just the same thing as humans because they do not fit into society as humans do, so the surrounding society must compensate for the way the computer fails to reproduce what it replaces. This means that a complex judgment is needed to test whether software fits well enough for the surrounding humans to happily ‘repair’ the differences between humans and machines. This is much more than a matter of deciding whether the cogs spin right.”

The Secret Life of Automation.pdf - 21

65

Secret:
Whatever is being automated,
it's not the *testing*.
And it's not the user, either.

The Secret Life of Automation.pdf - 39

66

Secret:

**People tend to focus on the How and the What of automated button-pushing, and on how it looks (it's dazzling!).
But they don't focus on the Why.**

The Secret Life of Automation.pdf - 50

67

Secret:

You can use tools to map, probe, visualize, stir, disrupt, amplify, randomize... and EXPLORE.

The Secret Life of Automation.pdf - 51

68

Exploration and Analysis to the rescue!

By exploring and analyzing the product (with the help of powerful tools), we may discover

- real problems that matter to real people
- fast feedback to help address them
- specific new risks to focus on
- “broken leg” problems to influence strategy
- obstacles that we could surmount with help
- radical shortcuts for exploring specific risks
- ways to produce useful maps of the product
- obstacles that we could surmount with help

Testability to maximize the power of testing AND checking.

The Secret Life of Automation.pdf - 52

69

Secret:

Testing is confused with confirmation, and confirmation is a problem.

The Secret Life of Automation.pdf - 57

70

On Confirmation

Most of the technology of “confirmatory” non-qualitative research in both the social and natural sciences is aimed at **preventing discovery**. When confirmatory research goes smoothly, **everything comes out precisely as expected**. Received theory is supported by one more example of its usefulness, and requires no change. As in everyday social life, **confirmation is exactly the absence of insight**. In science, as in life, dramatic new discoveries must almost by definition be accidental (“serendipitous”). Indeed, they occur only in consequence of some mistake.

Kirk, Jerome, and Miller, Marc L., Reliability and Validity in Qualitative Research (Qualitative Research Methods). Sage Publications, Inc, Thousand Oaks, CA, 1985.

The Secret Life of Automation.pdf - 58

71

Morals of these Stories

- Confirming the happy path often inadvertently focuses on *avoiding* finding problems...
- ...but if you want to find the banana peels that the customers will trip over, you’d better
 - map the product (and iterate)
 - *vary* your paths
 - look for *problems*, in products, tools, and checks
 - learn from *every* problem



72

Time for some research!

- To what degree are the checks and their outcomes being analyzed?
- What is the nature of the bugs that are being found using automated checks?
- What kinds of things might be a really good idea to check?
- To what degree is the checking effort and its value being analyzed?
- How long does it take to develop, run, interpret, debug, and maintain ?
- Who is doing the automated checking? Testers? Programmers? Both?
- How much are testers, check developers, toolsmiths and developers collaborating?
- Do checks really “allow testers more time for exploratory testing”?

It's probably not possible to make very many useful generalizations from my anecdotal and not-very-scientific research, but it might be possible to learn some useful things from stories.
Stories can help to reveal the Secret Life.

The Secret Life of Automation.pdf - 53

73

To what degree are checks and their outcomes being analyzed?

- In a very unscientific survey, my sources and my observation suggest “not very much” — especially not the ones that consistently produce green results.
 - Why? Perhaps because 160,000 “automated tests”, reviewed at a rate of 1000 per day, would take almost half a year to review completely.
- When they are being analyzed, I hear stories of
 - expensive and unhelpful formalization
 - busy-work
 - sunk cost bias

The Secret Life of Automation.pdf - 61

74

What is the nature of the bugs that are being found using automated checks?

- Since the checks and their outcomes are not being analyzed, the answer is unclear.
- From testers, several reports of bugs found during *development* of automated checks, but almost never thereafter.
- From developers using TDD, testing, checking, and development are all intertwined, so the answer here is unclear too.

The Secret Life of Automation.pdf - 62

75

What kinds of things might be a really good idea to check?

- Units
 - Developer checks (TDD style) are relatively cheap to develop and maintain, since check development is done in parallel with programming and testing.
 - Feedback from lowest-level checks is very fast; never goes beyond the developer's machine.
- Data
 - Says one source: "I have a set of data that has to get massaged from its raw form to be usable by the software. There are 10,000 input files. They're processed, rendered into a certain form, put into a data structure, indexed, and stored. If one of those files is corrupted, it will give you an error message. There are 100,000 users, and they will have a high probability of finding the problem. With a single tester, you don't have that. Thus you may want a tool to visit all 10,000 of those."
- Sanity Checks
 - Checks for the existence of elements of the product, without a ton of intelligence about their content (that's hard)

The Secret Life of Automation.pdf - 63

76

What is the role of roles?

- Who is doing the automated checking? Testers? Programmers? Both?
 - Lots of variation here, but at the higher levels, it seems to be testers.
 - Lots of variation in programming skill
- How much are testers, check developers, toolsmiths and developers collaborating?
 - Lots of stories that add up to “not enough”.
 - Lots of stories that suggest teams aren’t taking advantage of developer expertise
 - Lots

The Secret Life of Automation.pdf - 64

77

Analysis?

- To what degree is the checking effort and its value being analyzed?
 - Not much, so far as I can tell from working around the world, and from colleagues.
 - This is not surprising; people are not doing this for the rest of testing either.
- How long does it take to develop, run, interpret, debug, and maintain checks?
 - See above.
 - For all practical purposes, nobody is talking candidly and publicly about this.
 - Failures are happening; they *must* be happening. But no one has an incentive to talk about it, and there are lots of disincentives.

The Secret Life of Automation.pdf - 65

78

Do checks really “allow testers more time for exploratory testing”?

- Due to all the other secrets, this appears to be a secret too.
- There is by nature *a lot* of overhead associated with automated checking. It’s software development!
- There is little consensus that more ~~exploratory~~ testing is actually happening.

The Secret Life of Automation.pdf - 66

79

Ideas for Teams

- Development tasks *include* testing tasks.
 - The more you treat outsourcers as outsiders, the more likely their work will be misaligned with your work.
 - The more distance between “testers” and “automators”, the more likely that one won’t align with the other’s needs.
- Testing benefits from diversity and requisite variety. Diversify your team and your tactics.
- Learn from every bug.
- Revisit your checks; analyze their relevance.
- The belief that you **MUST** automate user-level checks suggests problems in your development process and your models of testing. Fix **THOSE** problems.

The Secret Life of Automation.pdf - 67

80

Secret:
There are no flaky checks.
But there *are* flaky interpretations of
results from checking.

The Secret Life of Automation.pdf - 70

81

Claim:
Tools can help us to do
more testing
faster
than we've ever done it before.

The Secret Life of Automation.pdf - 71

82

Secret:
Tools can help us to do
more **lousy, shallow** testing
faster **and worse**
than we've ever done it before.

The Secret Life of Automation.pdf - 72

83

Secret:
When there are “flaky” checks, *something*
becomes desensitized.

The Secret Life of Automation.pdf - 73

84

Secret:
**When programmers, testers, “automators”,
and toolsmiths are separated, secrets will
multiply.**

The Secret Life of Automation.pdf - 74

85

Secret:
**People who have both the tester and
builder mindsets are rare, and even they
find switching mindsets to be hard.**

The Secret Life of Automation.pdf - 75

86

Secret:
**Many people try to make testing
repetitious and over-focused,
and therefore fragile.**

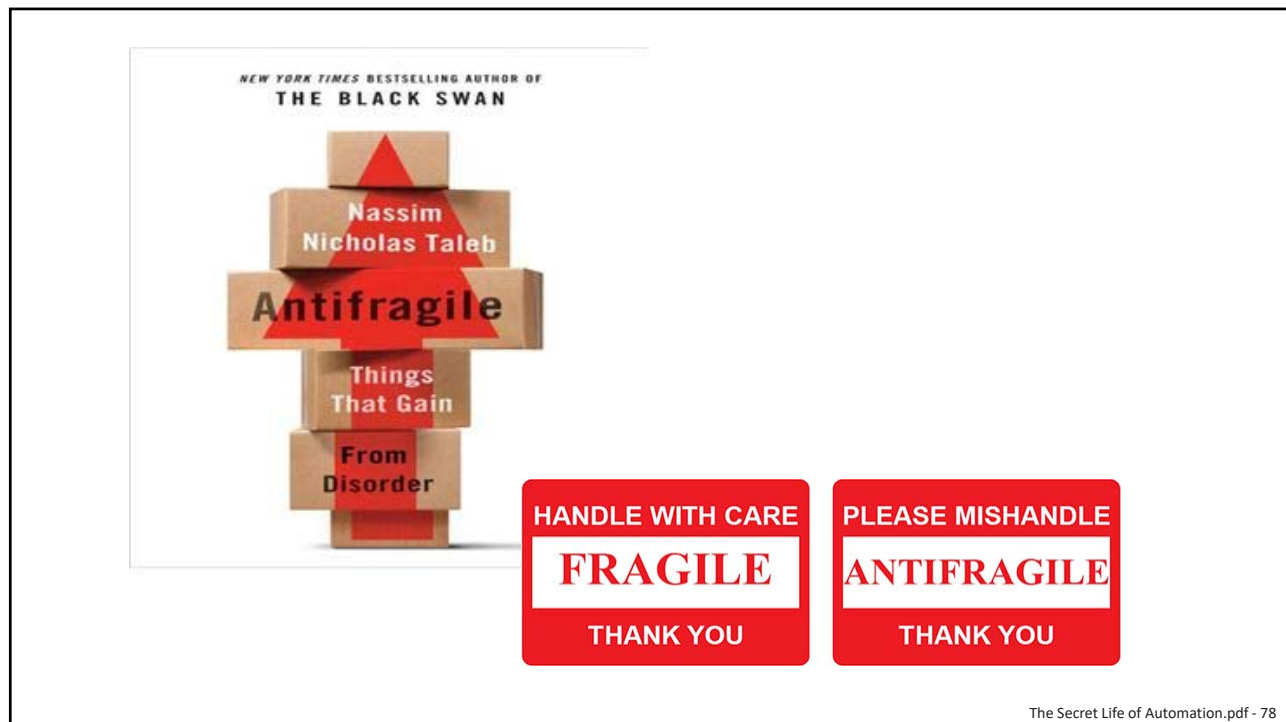
The Secret Life of Automation.pdf - 76

87

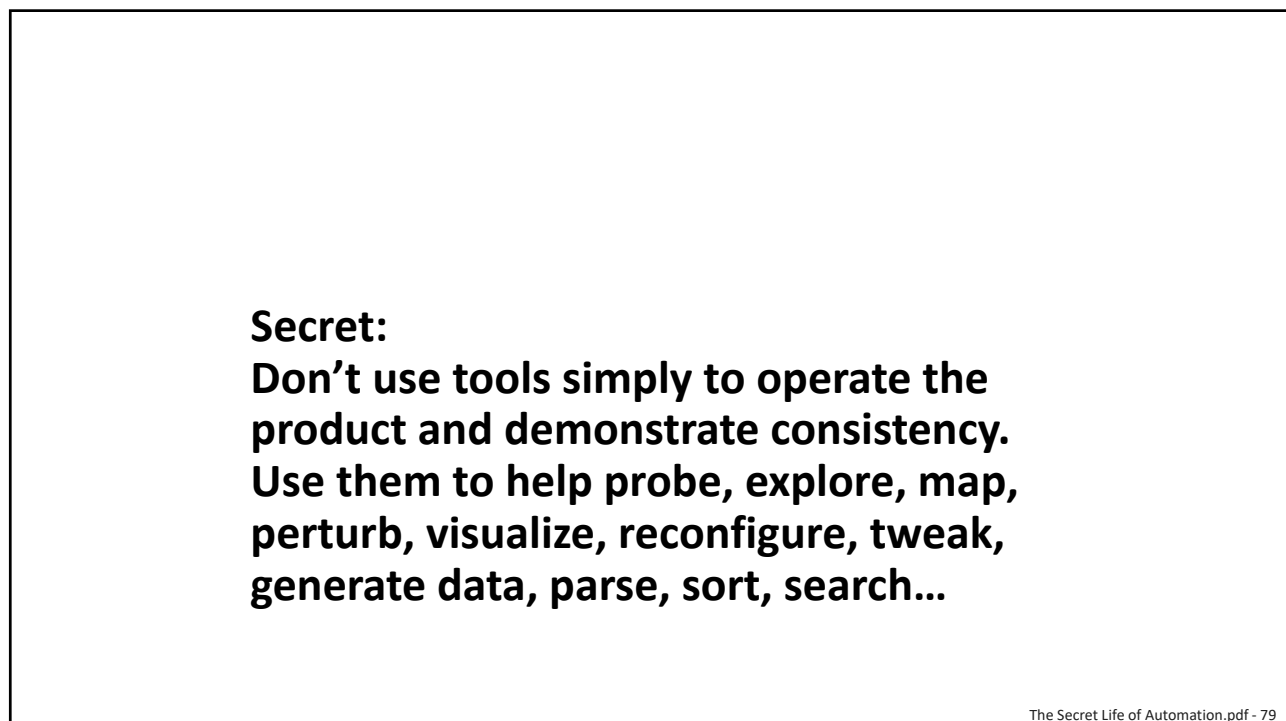
Secret:
**Successful testing should be an *antifragile*
and *antifragilizing* activity.**

The Secret Life of Automation.pdf - 77

88



89



90

Secret:
We don't know how to test until we've
***tried* to test.**
And we don't know how to apply tools
until we've *tried* to apply tools.

The Secret Life of Automation.pdf - 80

91

Secret:
Don't worry about building tools
knowing you'll throw them away. Take
advantage of the fact that as you're
***building* tools, you're testing.**

The Secret Life of Automation.pdf - 81

92

Claim:
Automated checking frees up more time for exploratory testing.

Secret:
Not necessarily. And certainly not when fixation on programming and maintaining automated checks dominates the testing effort.

The Secret Life of Automation.pdf - 82

93

Secret:
***Execution time* may be reduced, but there's also preparation, programming, debugging, troubleshooting, maintenance, analyzing failed checks, repair...**

The Secret Life of Automation.pdf - 83

94

Secret:

**Preparation, programming, debugging,
troubleshooting, maintenance, analyzing
failed checks, repair, etc.
may afford learning... but will it be about
things customers care about?**

The Secret Life of Automation.pdf - 84

95

Technology Rule 1:

**Any novel, non-trivial task will
take longer than you think it will.**

The Secret Life of Automation.pdf - 85

96

Technology Rule 2:
Rule 1 applies even after you've
taken Rule 1 into account.

The Secret Life of Automation.pdf - 86

97

Technology Rule 0:
There will be bugs.

The Secret Life of Automation.pdf - 87

98

Secret:
Too often, tool work becomes solution-
problemming.

The Secret Life of Automation.pdf - 88

99

Secret:
We shape our tools; thereafter they shape us.

—Marshall McLuhan

The Secret Life of Automation.pdf - 89

100

Secret:
You can't schedule epiphanies.
But you can make room for
epiphanies, and learn from every
one. Learn from every bug.

The Secret Life of Automation.pdf - 90

101

Secret:
The Program Manager wants to know
ONE THING above all: Are there
problems that threaten the on-time,
successful completion of the project?

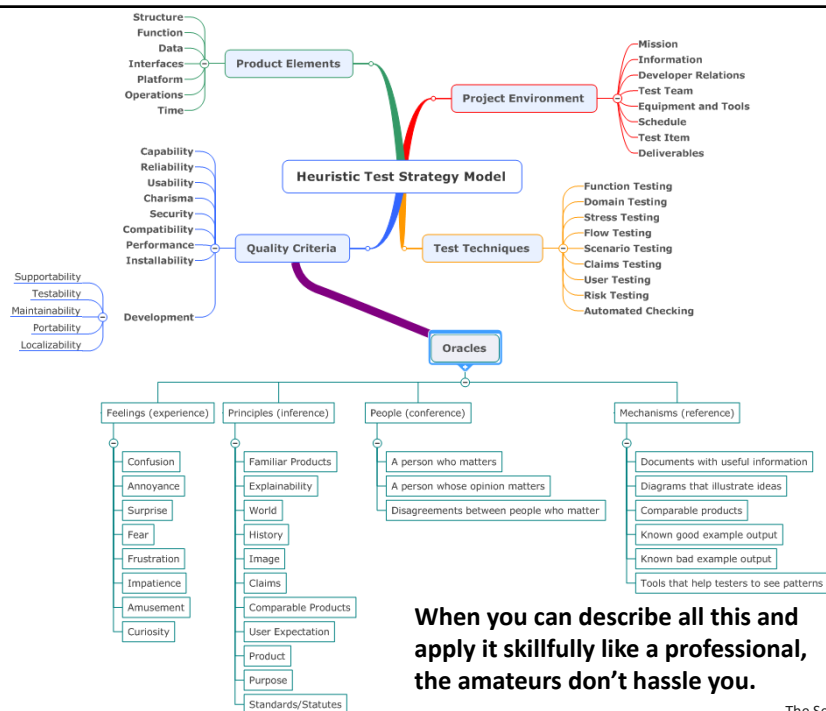
The Secret Life of Automation.pdf - 91

102

Secret:
When you diversify your coverage and risk models (and talk about them), you'll get less pressure to attempt and rely upon confirmatory checking.

The Secret Life of Automation.pdf - 92

103



When you can describe all this and apply it skillfully like a professional, the amateurs don't hassle you.

The Secret Life of Automation.pdf - 93

104

Thinking about Better Test Strategy

Try replacing...

Verify that...
Validate
Confirm that...
Show that it works
Pass vs. fail...
Executing test cases
Counting test cases
Automated testing
Test automation
Use cases

KPIs and KLOCs

With...

Challenge the belief that...
Investigate
Find problems with...
Discover where it *doesn't* work
Is there a problem here?
Performing experiments
Describing coverage
Programmed checking
Using tools in powerful ways
Use cases AND *misuse* cases AND *abuse* cases
AND *obtuse* cases...

Learning from every bug

The Secret Life of Automation.pdf - 94

105

More Stuff: Use The Google

- My blogs
 - "On Green"
 - "On Red"
- With James Bach
 - "A Context-Driven Approach to Automation in Testing"

The Secret Life of Automation.pdf - 95

106